VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

NÁVRH A REALIZACE UNIVERZÁLNÍHO ZÁZNAMNÍKU DAT

DESIGN AND IMPLEMENTATION OF A UNIVERSAL DATA LOGGER

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — TOMÁŠ VOKÁL
AUTHOR

VEDOUCÍ PRÁCE — ING. PAVEL HOUŠKA, PH.D.
SUPERVISOR

BRNO 2012

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

(na místo tohoto listu všijte originál a nebo kopii zadání Vaší práce)

ABSTRAKT

Tématem této práce je vytvoření univerzální záznamníku dat za použití .NET Micro Framework. Návrh byl otestován na vývojovém modulu EMX od společnosti GHI Electronics. Práce se nejdříve zaměřuje na seznámení čtenáře s .NET Micro Frameworkem a následně na popis architektury a ovládání dané aplikace.

ABSTRACT

Subject of this thesis is to design universal datalogger while using .NET Micro Framewrok. The design was tested on a developing kit EMX produced by GHI Electronics. At the beginning a problematic of .NET Micro Framework is introduced to a reader. After that architecture of project and mechanism of controlling device are described.

KLÍČOVÁ SLOVA

.NET Micro Framework, C#, záznamník dat, embedded zařízení, TinyCLR

KEYWORDS

.NET Micro Framework, C#, data logger, embedded devices, TinyCLR

PROHLÁŠENÍ

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a odbornou literaturu využitou při zpracovávání práce řádně cituji s uvedením úplného odkazu.

Tomáš Vokál

BIBLIOGRAFICKÁ CITACE

VOKÁL, T. *Návrh a realizace univerzálního záznamníku dat.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2012. 35 s. Vedoucí bakalářské práce Ing. Pavel Houška, Ph.D..

Obsah:

1 Úvod

Hlavním cílem práce je vyvinout záznamník dat (datalogger) s využitím nízkopříkonového hardwaru s grafickým dotykovým displejem a platformy .NET Micro Framework. Hlavní důraz je zde kladen na univerzálnost a autonomii zařízení, které je schopno fungovat zcela bez obsluhy.

Při vývoji programu byl kladen důraz na ovládání zařízení přes vestavěný dotykový displej, a také na možnost ovládat zařízení pomocí klientské aplikace přes rozhraní ethernet.

Cílem práce byla implementace rozhraní UART. Aplikace byla navržena abstraktně, aby pozdější implementace dalších běžně používaných průmyslových zařízení byla jednoduchá a kompatibilní se souborem funkcí celé aplikace.

Záznamník zde získává data z měřících jednotek připojených přes rozhraní UART. Získaná data průběžně zaznamenává do souborů na záznamové médium (jako záznamové médium zde byla zvolena SD karta) a nabízí uživateli možnost zobrazit data jak na displeji, tak i přes rozhraní ethernet pomocí WCF klienta.

Konfigurace zařízení je navržena tak, aby byla jednoduchá a uživatelsky přívětivá.

Hlavním důvodem zvolení .NET Micro Frameworku je hlavně moje motivace se tuto platformu naučit a vyzkoušet si vývoj na rozsáhlejším projektu.

Před představením samotného projektu vám budou představeny základy .NET Micro Frameworku a některé technické detaily o architektuře tohoto frameworku.

2 .NET Micro Framework

Tato kapitola je vytvořena na základě překladů a doplnění souvisejících zdrojů [1],[2],[3].

2.1 Co je to .NET Micro Framework?

Microsoft .NET Micro Framework je velice malé a efektivní .NET běhové prostředí. Je určené ke spouštění řízeného kódu[1] na zařízeních, která jsou příliš malá a svými parametry nesplňují požadavky pro použití Windows CE nebo .NET Compact Framework[2].

.NET Micro Framework především umožňuje uživateli psaní embedded[3] aplikací pro malé embedded zařízení za použití Visual Studia a C#. .NET Micro Framework, .NET Compact Framework a plný .NET Framework jsou si navzájem tak podobné, že umožňují uživateli používat stejné vývojové nástroje a jazyky, které se používají k vytváření desktopových aplikací a aplikací pro inteligentní zařízení (PDA a smartphony), pro vývoj aplikací pro mikrokontroléry. Značnou výhodou .NET Micro Frameworku je také možnost použití rozšiřitelného hardwarového emulátoru určeného především pro rychlé prototypování a ladění.

.NET Micro Framework ke svému běhu nepotřebuje základní operační systém nainstalovaný na použitém mikrokontroléru, nicméně i toto řešení je dostupné. Běh bez operačního systému je umožněn použitím zmenšené verze *Common Language Runtime* (TinyCLR) operující přímo na zvoleném hardwaru. Z tohoto důvodu je .NET Micro Framework často označován jako „*bootovatelné aplikační prostředí“*. Toto prostředí má především malý paměťový otisk (řádově několik set kilobytů RAM) a nevyžaduje, aby procesor měl jednotku správy paměti (MMU). Díky tomu může .NET Micro Framework běžet na malých a levných 32 bitových procesorech bez větších nároků na spotřebu energie.

2.2 Proč byl .NET Micro Framework vyvinut?

Tento oddíl popisuje, proč Microsoft vyvinul .NET Micro Framework. .NET Micro Framework nabízí především vývojáři „jiný přístup“ k vývoji aplikací. Než přejdeme k vysvětlení „jiného přístupu“, je nutno uvést, jak probíhal vývoj embedded zařízení v minulosti.

2.2.1 Vývoj embedded zařízení v minulosti

Vývoj aplikací pro embedded zařízení od nuly a jejich programování v C++, ANSI C nebo v assembleru může být (a většinou také je) náročné. Vývojáři embedded zařízení jsou zvyklí psát kód, který přímo ovládá zvolený hardware (neboli vývoj ovladačů a interface pro konkrétní hardware). Spuštění softwaru napsaného pro jeden mikrokontrolér pak přirozeně nebude fungovat na jiné platformě, i když CPU jádro je stejné. Každá deska má totiž jiné sběrnice, řadiče přerušení, paměť a I/O rozhraní.

---

[1] V angličtině označován jako „managed code“. Řízeným kódem se označuje kód, který potřebuje ke svému vykonání CLR. Takový kód je pouze souborem instrukcí, nikoliv nativním kódem. Dále budu uvádět pouze „řízený kód“.

[2] Windows CE je dnes již stará verze embedované distribuce Windows. Nicméně jak Windows CE, tak .NET Compact Framework mají zajištěnou podporu do roku 2022. Z důvodu nejlepší výstižnosti byly uvedeny jako referenční právě tyto platformy.

[3] „Embedded“ je v češtině též označováno jako „vestavěný“, či „zabudovaný“. Jelikož se jedná o zavedený pojem, budu dále používat jeho anglické pojmenování. Obvykle se tak označuje software

Nástroje a vývojová prostředí pro vývoj tohoto druhu aplikací nejsou většinou snadno použitelné a komplexní. Každý dodavatel CPU poskytuje zpravidla vlastní kompilátor a vývojové nástroje, což ztěžuje simulaci hardwaru a ladění embedded aplikace ve vývojovém prostředí na PC.

2.2.2 Jiný přístup k vývoji

Standardní hardwarové platformy s .NET Micro Frameworkem již od základu nabízejí jiný přístup. Ten je zaměřen především na vývojáře a snaží se odstranit veškeré překážky ze světa hardwaru a umožňuje vývojáři soustředit svou pozornost převážně na aplikační vrstvu.

Hardware určený pro .NET Micro Framework je zpravidla mikrořadič připravený k okamžitému použití. Obsahuje většinu standardních hardwarových komponent jako jsou paměť, vstupní/výstupní (GPIO) porty, sériové porty a displej pro okamžité ovládání (displej není mandatorní součást a zpravidla se nedodává standardně, ačkoliv funguje okamžitě po zapojení k vývojové desce). Kód pro komunikaci s hardwarovými komponenty je již od základu připravený a tudíž uživateli stačí pouze napsat aplikaci pro ovládání těchto komponent. Výsledkem zmíněného přístupu je možnost soustředění se pouze na problémy spojené s aplikační vrstvou.

.NET Micro Framework abstrahuje přístup k hardwaru skrze své základní knihovny. Výsledkem této abstrakce je přístup k hardwarovým komponentům jako k objektům. Zmíněné pojetí nám od počátku umožní programovat objektově orientovanou aplikaci, což programování skrze C++, ANSI C nebo assembler neumožňuje. Místo toho, abychom se zabývali detaily hardwaru a nastavování bitové masky pro konfiguraci periferního hardwaru, stačí nastavit všechny vlastnosti jednoduše na požadovaném objektu. Tento přístup k problému bývá často označován jako „*managed driver*“[4] a díky němu je embedded aplikace nezávislá na konkrétní platformě.

.NET Micro Framework se programuje za pomocí jazyků C# nebo Visual Basic (od verze 4.1) ve vývojovém prostředí Microsoft Visual Studia (je možno programovat i v Microsoft Visual Studiu Express[5]). Pro uživatele, který už někdy programoval nějakou .NET aplikaci pro stolní počítače nebo smartphony, bude vývoj embedded zařízení velice snadný a rychle pochopitelný.

Díky použití C#, který je moderním vysokoúrovňovým programovacím jazykem, je psaní kódu rychlé a čisté. Hlavní výhodou je také to, že kód je znovupoužitelný. Stejně jako u plného .NET Frameworku a .NET Compact Frameworku, .NET Micro Framework běží jako řízený kód. Kompilátor pro jazyk C# generuje procesorově nezávislý soubor instrukcí, který TinyCLR následně provádí na daném zařízení.

2.3 Řízený kód

Jak již bylo uvedeno, nedílnou součástí .NET Micro Frameworku je TinyCLR[6]. Kód, který je vykonávaný a řízený skrze CLR, se označuje jako řízený kód, zatímco kód, který se nezaměřuje na CLR, je znám jako kód neřízený (nativní). CLR vykonává výše zmíněný soubor instrukcí a poskytuje následující služby a výhody s nimi spojené.

[4] Neboli také řízené ovladače.
[5] Tato výhoda je rozepsána v oddílu Licencování
[6] Jedná se o zmenšenou a optimalizovanou verzi CLR. Zásady uvedené níže platí i pro jeho plnou verzi, proto bude v následujícím textu uvedeno pouze CLR.

2.3.1 Základní služby a výhody řízeného kódu:

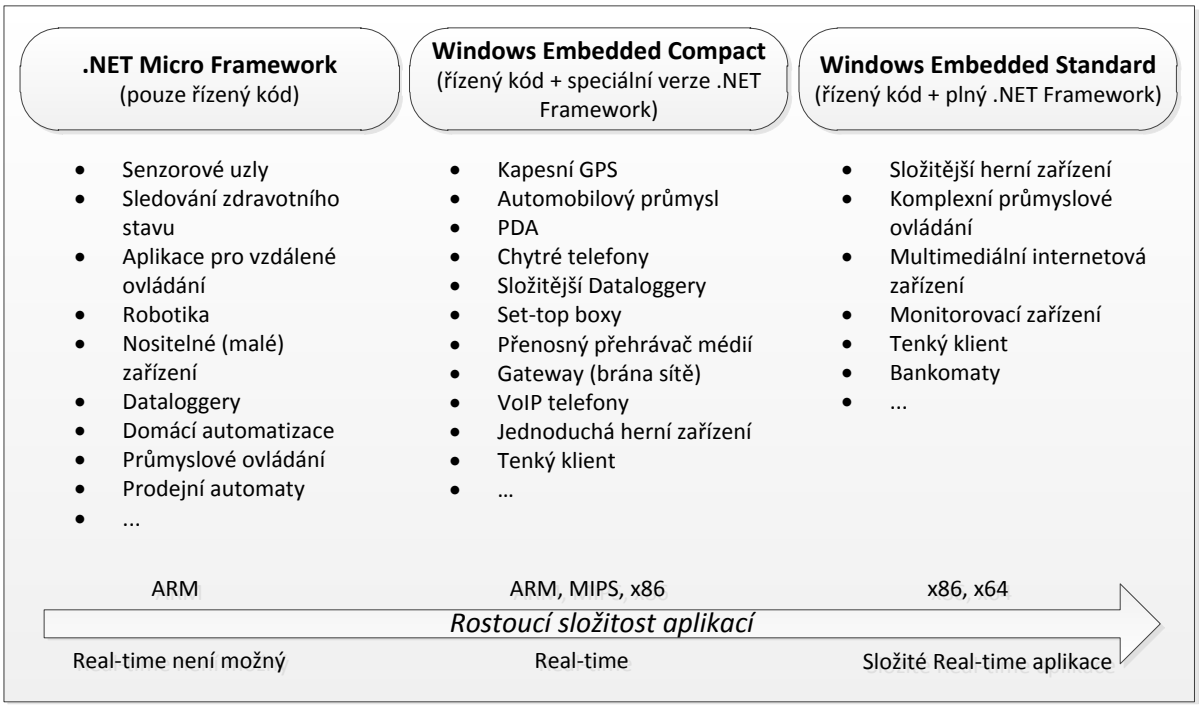
- Automatická správa paměti pomocí garbage collectoru
- Správa vláken a jejich synchronizace
- Zpracování výjimek
- Striktní kontrola datových typů
- Bezpečné a robustní řízení kódu
- Služby pro ladění kódu

CLR používá *garbage collector*, který automaticky uvolňuje nepoužívané paměťové bloky a řídí vlákna tím, že poskytuje výpočetní čas jednotlivým vláknům a metodám za účelem synchronizace přístupu ke sdíleným zdrojům. Řízený kód se provádí pod přísnou kontrolou CLR, která spravuje reference objektů na paměťové bloky. Nemusíme se tudíž zabývat používáním nebezpečných *pointerů*[7], které jsou spojené s nativním programováním.

S řízeným kódem nemůžeme přistupovat k paměťovým blokům poté, co byly objekty (či struktury) na ně ukazující zničeny. CLR uvolní objekty (struktury) pouze tehdy, když už nejsou referovány žádnou součástí programu. To také vynucuje přísnou typovou bezpečnost, neboť CLR brání provádění nebezpečného a nesystémového nativního kódu. Velkou výhodou řízeného kódu je také snadné a efektivní zpracování výjimek. Řízený kód je díky výše zmíněnému velice bezpečný a robustní.

Sestavy, sestavené z řízeného kódu, obsahují mnoho informací (metadata[8]) o sobě samých. Proto používání řízeného kódu umožňuje používat nástroje statické analýzy.

2.4 Pro jaké aplikace je .NET Micro Framework vhodný?



Obr. 1 Vhodnost technologií pro dané typy aplikací

[7] Neboli ukazatelů
[8] Neboli data o datech

Tento diagram představuje vhodnost použití embedded systémů a distribucí od společnosti Microsoft pro různé typy aplikací. Je nutné si uvědomit, že zde neexistuje žádná jasná hranice toho, kdy se aplikace hodí pro daný systém, nebo zda je možno využít jednoduššího systému.

Jak je možné vidět z uvedeného diagramu, je .NET Micro Framework výhodný především pro aplikace s velice omezenými nároky na výpočetní výkon a paměťové zdroje, které jsou provozovány na jednoduchých a levných zařízeních.

.NET Micro Framework je prozatím nejmenší .NET platformou, která poskytuje pouze podmnožinu z celkové funkcionality .NET Frameworku. .NET Micro Framework běží přímo na konkrétním hardwaru a vyžaduje jen asi 250 KB RAM. S každou novou generací tohoto frameworku se hardwarové nároky snižují.

.NET Micro Framework vyžaduje 32bitový procesor s architekturou ARM. Potenciální nevýhodou se zde může jevit nemožnost použít 16bitové či 8bitové procesory. Vhledem k zlepšování výrobních technologií a spotřeby zařízení jsou zde rozdíly minimální. Díky tomu, že .NET Micro Framework nevyžaduje MMU, může běžet na procesorech levnějších než kterákoliv distribuce Windows Embedded.

2.5 Omezení .NET Micro Frameworku

.NET Micro Framework není *real-time systém*[9]. Ačkoliv je navrhnut, aby běžel velice rychle, což je dostačující pro většinu aplikací, nemůžeme od něj čekat přesné real-time deterministické chování. Díky použití řízeného kódu se může lišit spouštění časovačů až v řádu jednotek milisekund. Také přerušení musí být nejdříve zpracována v běhovém prostředí a až poté (po několika milisekundách) jsou volány služby a metody určené ke zpracování těchto přerušení.

Navíc musíme počítat s tím, že o uvolňování paměti se stará pouze *garbage collector*. Pokud má zařízení málo paměti, *garbage collector* má právo přerušit všechna vlákna dokud není potřebná zásoba zdrojů uvolněna.

Také musíme počítat s tím, že vykonání řízeného kódu je díky potřebné režii trochu pomalejší než vykonávání kódu nativního. Instrukce řízeného kódu jsou zde kompilovány až při potřebě jejich spuštění. Neexistují zde žádná přímá volání nativního kódu, jelikož aplikace není kompilována do nativního kódu, nýbrž pouze do souboru instrukcí.

2.6 Licencování

.NET Micro Framework SDK je zdarma. Stejně tak je zdarma i nástroj Microsoft Visual Studio C# Express, který je základním vstupním vývojovým prostředím pro produkty Microsoftu. Použití Microsoft Visual Studia v edici Express je zdarma jak pro nekomerční, tak i komerční účely.

Celý .NET Micro Framework je vydaný pod freewarovou licencí Apache 2.0, což zaručuje jeho bezplatné používání i pro komerční účely. Každý uživatel si poté může díky nástroji Micro Framework Porting Kit zkompilovat Micro Framework na své zařízení (pro netypické chipsety je tento proces obtížnější a vyžaduje znalosti hardwaru a Micro Frameworku). Všechny zdrojové soubory nutné pro *port*[10] Frameworku i zdrojové soubory .NET Micro Frameworku, jsou uživateli volně přístupné a to zcela zdarma.

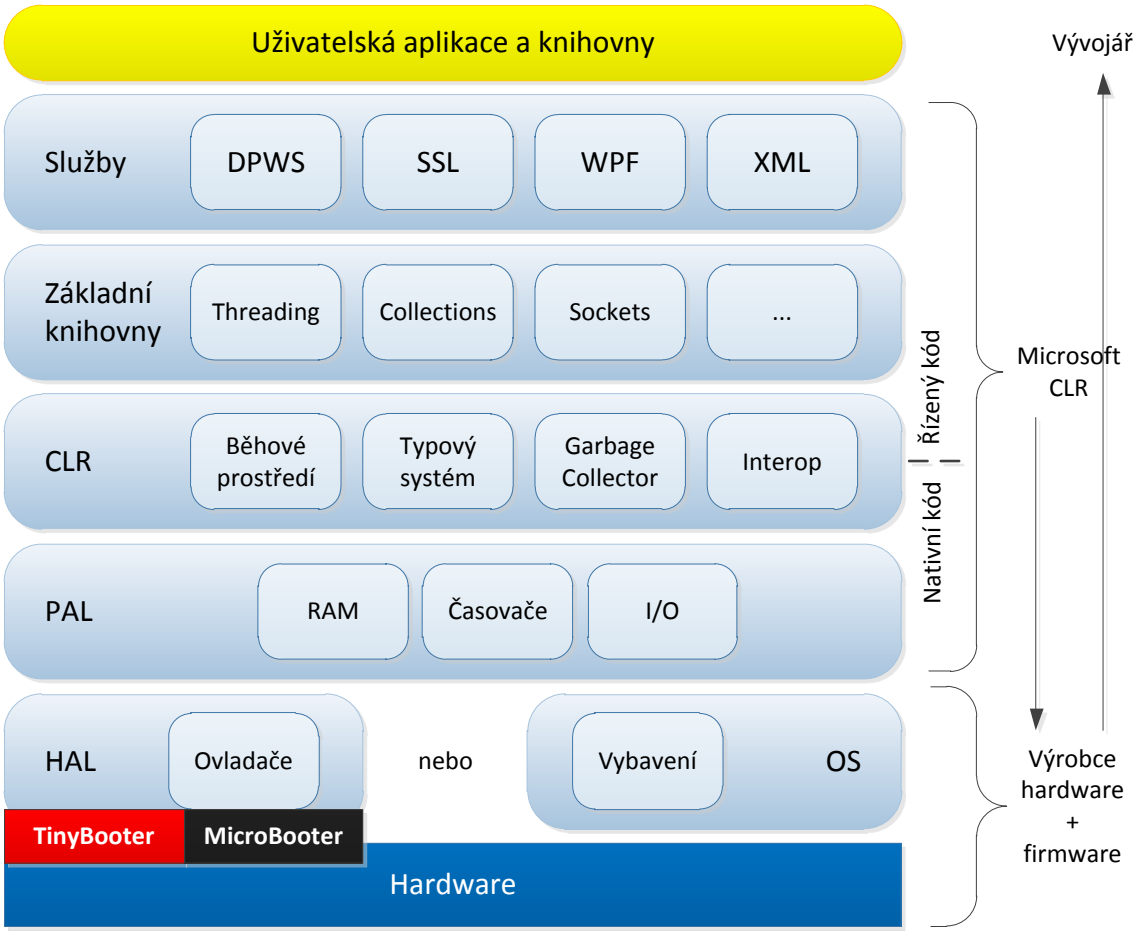
[9] Systém poskytující deterministické zpracování aplikací v reálném čase.
[10] Zkompilování Micro Frameworku na dané zařízení

2.7 Základní přehled výhod používání .NET Micro Framework

- Nízká cena vývoje (programátor nepotřebuje mít tolik znalostí jako vývojář nativního kódu)
- Rychlost vývoje (díky možnosti programovat pouze aplikační vrstvu se rapidně zrychlil vývoj)
- Volná technologie (jediným vstupním nákladem pro vývoj je zde pouze zkušební deska)
- Možnost okamžitého a snadného ladění kódu přímo na zkušebním zařízení
- Podobnost s .NET Frameworkem (jak již bylo zmíněno, každý .NET vývojář může být se svými znalostmi také vývojářem embedded zařízení)
- Veliké množství základních knihoven
- Možnost opakovatelné použitelnosti kódu (napsaný kód není vázán na určitou konfiguraci hardwaru)
- Přehlednost a rychlá správa kódu (Díky použití C# nebo Visual Basic a základních knihoven je kód velice přehledný jak pro samotného vývojáře, tak pro vývojáře třetích stran. Zároveň objektové programování zrychluje vývoj dokumentace.)
- Micro Framework plně podporuje všechny různé typy síťové komunikace

2.8 Technický přehled

Následující diagram představuje architekturu .NET Micro Frameworku. Důležitá je zde především hranice mezi řízeným a nativním kódem, která představuje hranici mezi dvěma programovacími jazyky. V našem případě C# pro řízený kód a C++ pro kód nativní.



Obr. 2 Architektura .NET Micro Frameworku

V následujících několika odstavcích bude popsána architektura ilustrovaná na (Obr. 2). Popis bude uveden od bootovatelné části směrem k uživatelské aplikaci.

2.8.1 Bootovatelné běhové prostředí: HAL, PAL

.NET Micro Framework poskytuje většinu služeb, které by uživatel čekal od klasického operačního systému. Z nich za zmínění stojí inicializace běhového prostředí, správa přerušení, řízení vláken a procesů, řízení haldy a další podpůrné funkce potřebné pro běh aplikací. Výše zmíněné funkce umožňují běh .NET Micro Frameworku přímo na hardwaru bez nutnosti operačního systému. Je zde ovšem i možnost fungování Micro Frameworku nad operačním systémem, pokud je potřeba. Často se této možnosti využívá při požadavcích real-time aplikací, kdy se nízkoúrovňový systém stará o správu a přístup k real-time zdrojům a .NET Micro Framework o vyšší řízení.

Základní vrstvou .NET Micro Frameworku je hardwarová abstraktní vrstva (HAL), jejíž velikost se pohybuje mezi 20-30 kB. Tato vrstva je jedinou částí .NET Micro Frameworku, která je přímo závislá na zvoleném hardwaru a tvoří se pomocí tzv. "portu". HAL poskytuje infrastrukturu pro zavedení aplikace a abstrahuje přístup k periferiím a hardwaru pouze na funkční volání. (Pokud .NET Micro Framework běží nad již nabootovaným operačním systémem, HAL poskytuje stejnou funkcionalitu skrze volání funkcí daného operačního systému.)

Klasické funkce operačního systému jsou v Micro Frameworku rozděleny mezi HAL a Common Language Runtime (CLR). HAL poskytuje rozhraní přístupu k hardwaru (například IRQ, časovače, a I/O), ale nemá plný *kernel*[11], aby sám o sobě mohl poskytovat řízení procesů a vláken nebo jakékoliv haldy. Pro tyto funkce je určeno CLR.

HAL kernel podporuje dva exekutivní režimy: jednovláknové aplikace a interrupt service routine (ISR). Je důležité si uvědomit, že HAL nemá žádný plánovač instrukcí. HAL je pouze jednovláknová operace, nad kterou běží CLR. Všechny uživatelské aplikace se vykonávají v CLR, které se poté stará o efektivní přerozdělení výpočetního času mezi tyto aplikace.

Dalším klíčovým požadavkem konstrukce HAL vrstvy je nízké využití procesoru pro snížení spotřeby energie, na které je značně optimalizována.

Vrstva nad vrstvou HAL je vrstva nazývaná Platform Abstraction Layer (PAL), která abstrahuje časovače, paměťové bloky, asynchronní komunikaci, seznamy a další datové struktury pro CLR, jež už nabízejí strojově (nativní) zaměřený kód HAL jako řízený kód pro aplikační knihovny.

2.8.2 .NET Micro Framework CLR

.NET Micro Framework CLR je podmnožinou .NET Framework CLR, které je běhovým prostředím pro plný .NET Framework. .NET Micro Framework CLR poskytuje robustní aplikační podporu. Jak již bylo zmíněno v předcházejícím oddílu, CLR spravuje paměť, provádění aplikačních vláken a kódu a další systémové služby.

TinyCLR je navíc schopno poskytnout všechny tyto funkce a služby při zachování velice malého paměťového otisku. Pokud jsou použité všechny dostupné funkce, TinyCLR zabírá pouze 390 kB paměti. Vaše TinyCLR implementace může být menší či větší než tento odhad v závislosti na konkrétním hardwaru, který používáte.

[11] Neboli jádro operačního systému

Běh TinyCLR je velmi rychlý. Přibližná měření ukazují, že je schopno obhospodařit kolem 15000 funkčních volání za sekundu při taktu 27,6 MHz.

Hlavní cíle při návrhu TinyCLR byly:

- Minimální paměťový otisk
- Možnost běhu bez operačního systému
- Možnost ho spouštět z ROM nebo flash paměti
- Stejně jako HAL je optimalizován pro maximální efektivity a šetření energie.

Vzhledem k výše zmíněným cílům bohužel TinyCLR obsahuje mnoho omezení. Především pak nepodporuje generické typy a kolekce (s výjimkou ArrayList).

2.8.3 Základní knihovna tříd + služby

Knihovny základních tříd jsou podobné jako je to u .NET Frameworku. .NET Micro Framework zde volí jiné postupy a řešení jejich implementací i když se jmenují stejně. Dále je nutno poznamenat, že je zde rozdíl v knihovně mscorelib.dll, která se stará o reprezentaci objektu jako takového, takže není možno odkazovat knihovny .NET Micro Frameworku v .NET Frameworku a naopak.

Knihovny jsou dále upraveny především po potřebu vývojáře embedded zařízení. Zejména jsou zde knihovny určené pro komunikaci (jak po většině typů průmyslových sběrnic, tak i po ethernetu), což je asi jednou z nejsilnějších stránek .NET Micro Frameworku.

2.8.4 Aplikační vrstva

Aplikační vrstva obsahuje uživatelem napsaný kód, který přes základní knihovny a služby ovládá programovaný hardware. V současné době existují 4 základní možnosti, jak ovládat program:

- Pomocí vstupů GPIO a tlačítek na nich napojených
- Pomocí dotykového displeje
- Pomocí webového serveru, který je možno na zařízení spustit
- Pomocí DPWS (jedná se o zúženou funkcionalitu WCF), které nám umožňuje ze zařízení udělat server či klienta a pomocí předem určeného datového kontraktu vyměňovat a upravovat požadované objekty.

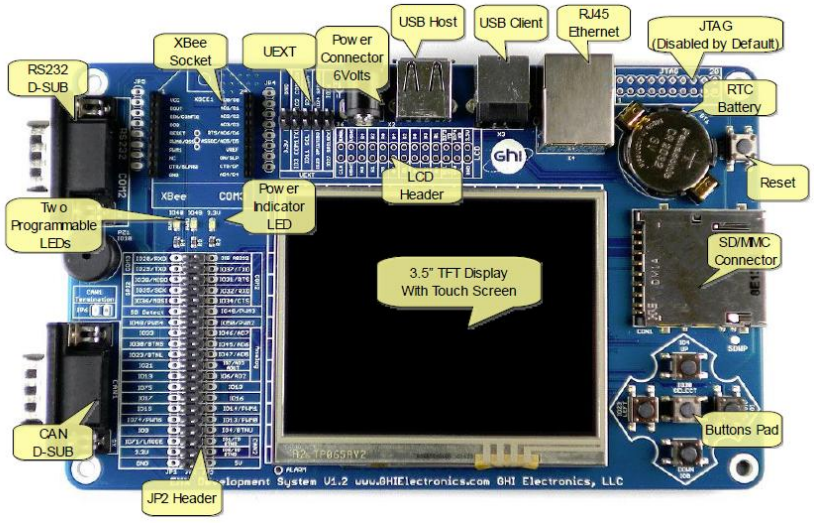
3 Zkušební zařízení

K ladění aplikace byl použit zkušební modul EMX od společnost GHI Electronics, který byl za účelem testování zakoupen. Modul obsahuje všechny dostupné porty a konektory, které byly k vývoji aplikace nezbytné.

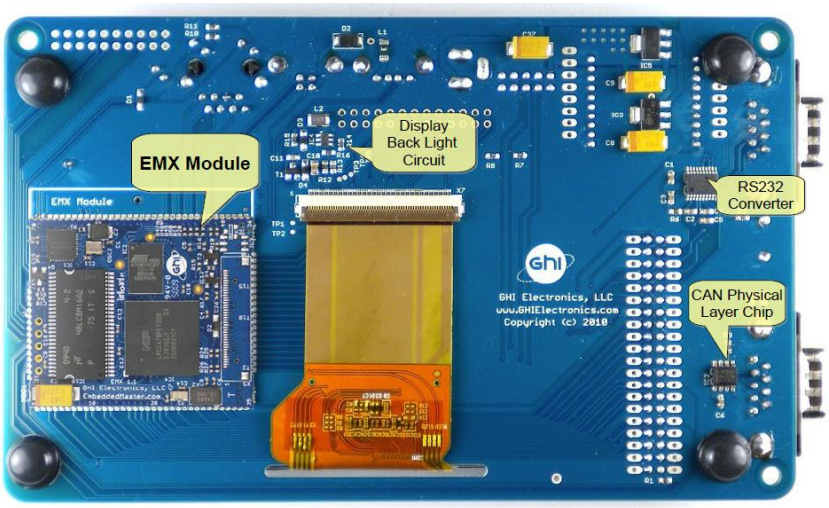
Výbavu a parametry tohoto modulu jde označit jako nadstandardní. Výsledná aplikace je navržená tak, aby fungovala na zařízeních s podstatně menším výkonem a vybavením. Ke všem modulům a procesorům od společnosti GHI Electronics jsou také dodávána rozšíření základních knihoven .NET Micro Frameworku, kterých bylo při vývoji aplikace využito.

Základní parametry zařízení:

- Procesor - 72 MHz 32bit ARM 7
- 16 MB RAM a 4.5 MB FLASH
- 320 x 240 3.5" TFT displej s dotykovým ovládáním
- RJ-45 Ethernet konektor
- 2x sběrnice SPI Master (8/16bit)
- I2C rozhraní
- 4x UART (sériový port), 1x rozhraní RS232 s podporou handshake
- 2x rozhraní CAN
- 6x PWM signály
- Podpora sběrnicových rozhraní
- Slot pro SD/MMC kartu
- USB klientský port
- USB Host port
- Záložní baterie pro hodiny
- LED diody a tlačítka
- Zabudovaný Piezo senzor
- Možnost napájení přes USB nebo DC (vstupní napájení je 6 voltů přes 2.1 mm konektor)
- Soupis všech parametrů a schéma zapojení je uveden v [4].



Obr. 3 Vývojový modul EMX, přední pohled [4].



Obr. 4 Vývojový modul EMX, zadní pohled [4]

4 Realizace programu záznamníku dat

V této kapitole jsou popsány základní segmenty realizace aplikace. Aplikace je pouze jednovláknová, což je pro účely dataloggeru dostačující. Realizace aplikace byla provedena v programovacím jazyku C# a ve vývojovém prostředí Microsoft Visual Studio 2010 Professional.

K vývoji aplikace bylo hojně využíváno knihoven GHI Electronics dodávaných s výše zmíněným vývojovým modulem. Tyto knihovny nám především poskytly vylepšenou kontrolu SD karty a souborového systému.

Dále bylo využito knihoven grafického rozhraní od téže společnosti, které byly pro potřeby aplikace značně upraveny a doplněny o chybějící potřebné komponenty.

4.1 Úvod

4.1.1 Co je to datalogger (záznamník dat)

Datalogger je obecně zařízení sloužící ke sběru a ukládání analogových či digitálních dat. Jeho hlavním úkolem je zaznamenávat data v čase a podle místa jejich vzniku.

Zařízení tohoto typu obvykle získává data ze svých periférií a ty schématicky shromažďuje pro pozdější zpracování. Je zde kladen hlavně důraz na schematičnost dat a jejich přesné a rychlé zaznamenávání v čase. Jako formát záznamu dat se zde využívají nejčastěji soubory typů „csv“ nebo „xml“, které jsou lehce strojově překladatelné. U souborů typu „xml“ můžeme z hodnoty sestavit tzv. „NoSQL[12]“ databázi.

Dalším klíčovým požadavkem dataloggeru bývá jeho autonomie. Jsou to často zařízení umístěná ve špatných provozních podmínkách průmyslových podniků, kde jsou podmínky provozu velice špatné. Tato zařízení musí být schopna běžet 24 hodin v kuse, třeba i několik měsíců v kuse, a neustále shromažďovat data pro pozdější analýzu. Bezobslužný provoz tohoto typu zařízení je jednou z klíčových podmínek.

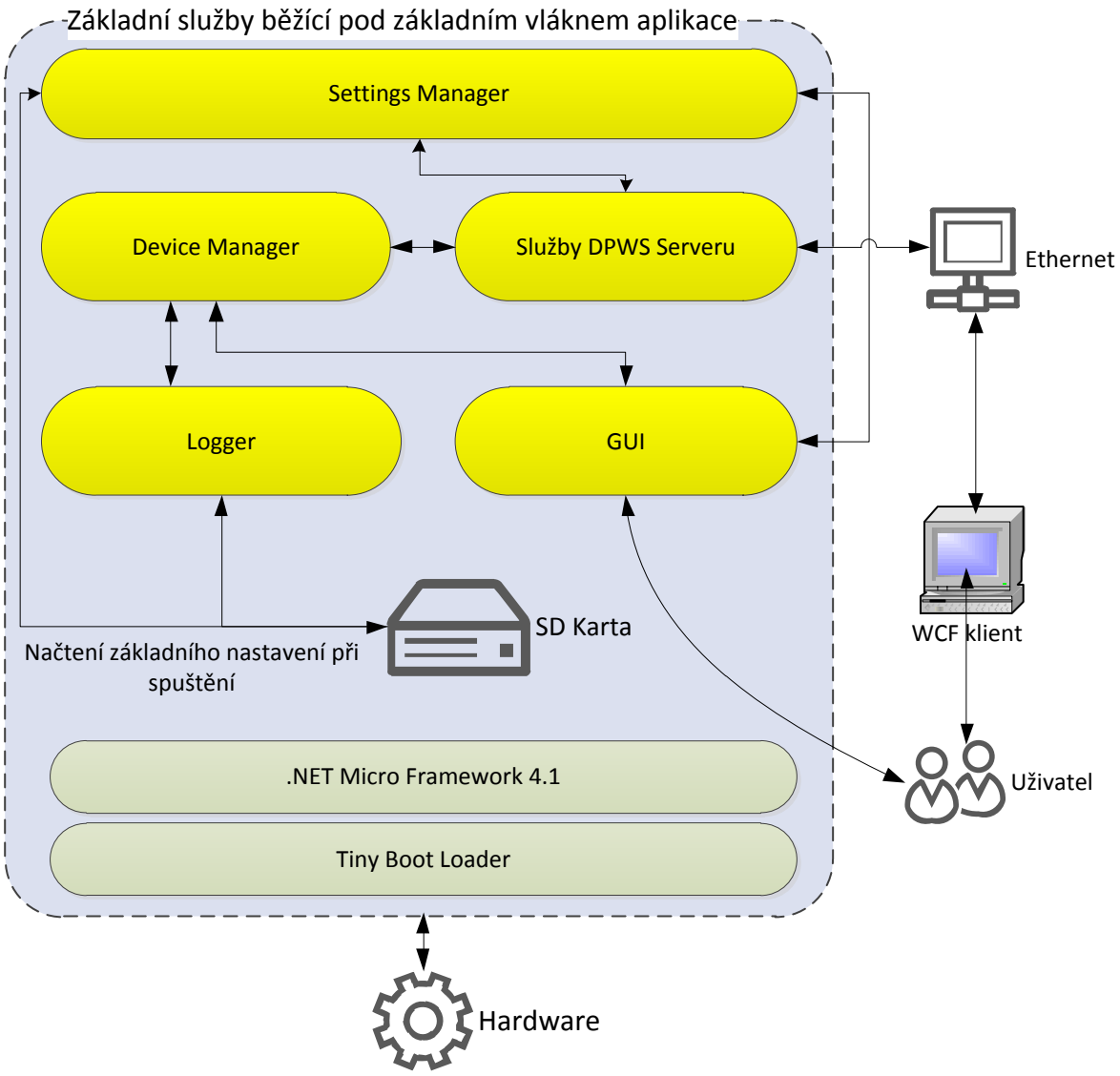
4.1.2 Proč byl zvolen .NET Micro Framework

.NET Micro Framework je především důležitou součástí strategie pro vývoj embedded zařízení společnosti Microsoft. Principy využité v této platformě jsou aplikovatelné i na vývoj mobilních zařízení a zařízení s operačním systémem Windows Embedded. .NET Micro Framework mi také pomohl vyřešit nutnost psát ovladače k jednotlivým periferiím a umožnil mi vyvíjet kód, který bude použit u dalších projektů. Z důvodu toho, aby se kód dal v budoucnosti použít a rozšířit, byla zvolena určitá řešení, která budou představena dále v této kapitole.

[12] Jako NoSQL databáze se označuje každý strukturovaný soubor dat se strukturou databáze, která není založená na SQL dotazech. Tyto databáze se používají hlavně na místech, kde je použití SQL serveru nemožné nebo nevhodné.

4.2 Základní architektura projektu

Následující diagram představuje základní schéma projektu. Jak již bylo zmíněno výše, celá aplikace se skládá pouze z jednoho jádra. Zmíněný diagram tedy obsahuje služby spouštěné v hlavním aplikačním vláknu.



Obr. 5 Základní architektura projektu.

Hlavní částí programu je především těchto 5 služeb:

- Settings Manager
- Device Manager
- Logger
- WPF grafické rozhraní pro dotykový panel
- DPWS Server

4.2.1 Settings Manager

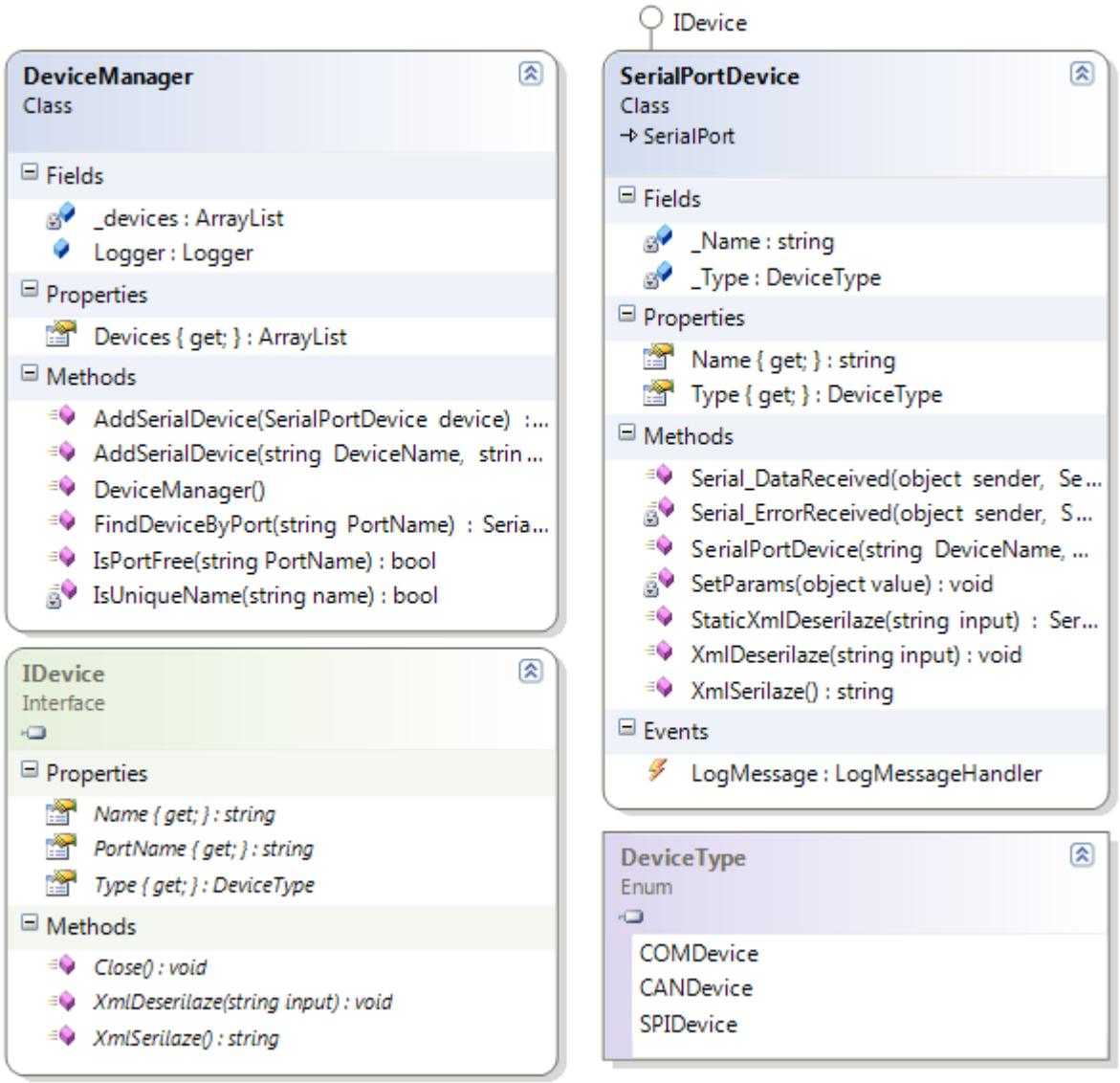
Settings manager je pouze jednoduchý objekt základního nastavení programu. Je deklarován jako static, aby byl přístupný všem třídám a metodám v dané aplikační doméně.

Je první službou, která se spouští v aplikaci. Jeho inicializace probíhá pomocí statické metody „init“, kdy přečte XML soubor nastavení přítomný na vložené SD kartě. Pokud je soubor poškozený nebo není přístupný, je pak načteno základní nastavení.

Pomocí souboru settings.xml tedy můžu nastavit celou aplikaci před jejím spuštěním a vše se mi nastaví podle potřeby při startu aplikace.

4.2.2 Device Manager

Device Manager je obecný správce zařízení. Jeho hlavní starostí je správa, inicializace a údržba periferií (především pak definice a nastavení komunikačních rozhraní). Device Manager je připraven k implementaci všech komunikačních rozhraní, které .NET Micro Framework nabízí díky tomu, že pracuje pouze s interface IDevice. V době psaní této práce je zhotovena pouze implementace UART rozhraní (sériový port).



Obr. 6 Diagram tříd - Device Manager

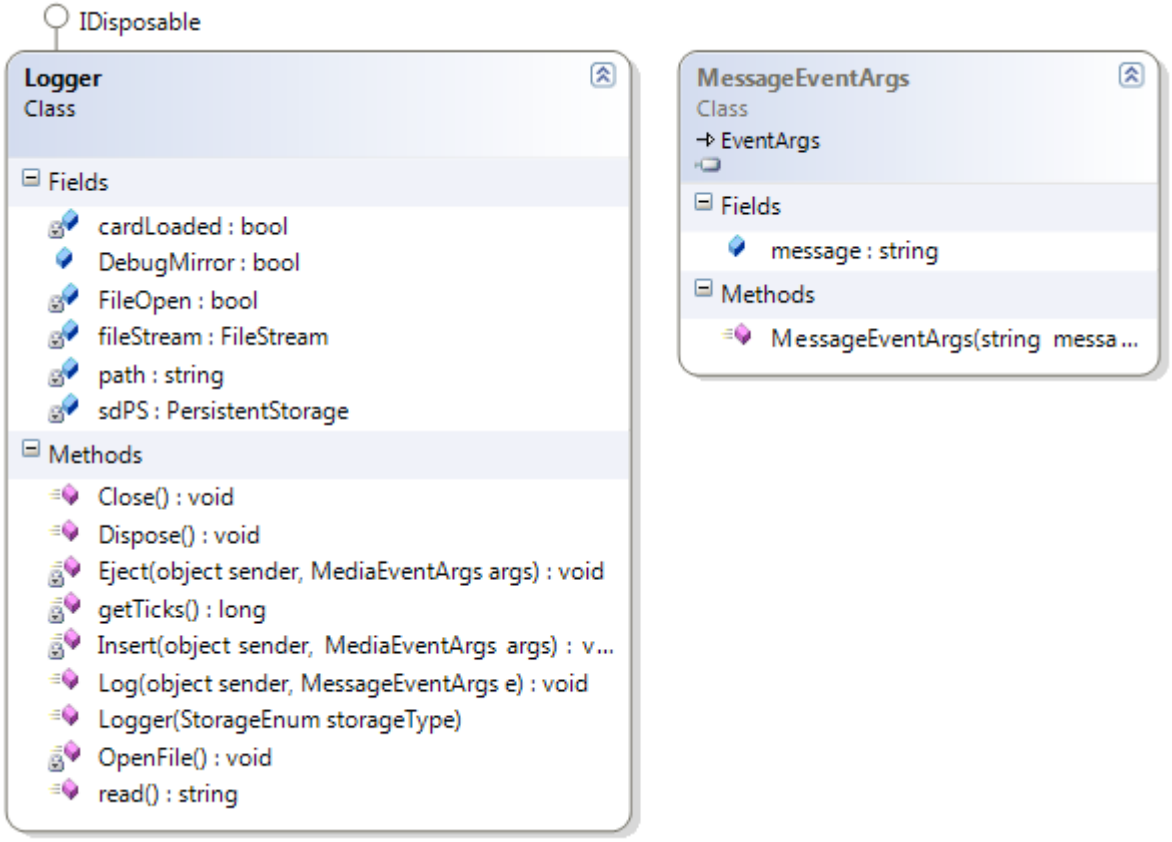
IDevice obsahuje základní metody pro obsluhu přístupu k danému rozhraní a také společné metody důležité pro každé rozhraní.

Jak je možno vidět z Diagramu tříd, Device Manager obsahuje definici pro přidání rozhraní pro sériový port. Po přidání tohoto zařízení se zavolá jeho konstruktor, který mu přiřadí událost log, která je poté volána z kódu zařízení a zaručuje spouštění zaznamenávání zpráv ve službě Logger v požadovaný čas.

4.2.3 Logger

Logger je služba sloužící k zaznamenávání dat. Je spuštěná po instanci Device Manageru, protože služba operuje na vloženou SD kartou. Jedná se o jednoduchou třídu, která tvoří po řádcích výstupní XML soubor. Původní řešení se souborem typu CSV bylo zavrhnuto.

Použití souboru typu XML nám nabízí lepší a typově bezpečné strojové zpracování a také soubor odolný vůči chybám.



Obr. 7 Diagram tříd – Logger

Ve svém konstruktoru se datalogger snaží otevřít soubor XML pro ukládání dat. Pokud soubor není na SD kartě přítomen, vytvoří se nový.

Pokud není přítomna SD karta, Logger ponechává dočasně všechna data v paměti, dokud uživatel SD kartu nepřipojí. O připojení a odpojení SD karty nás informují metody „Insert“ a „Eject“, které zpracují události vyvolané při zapojení a odpojení SD karty.

4.2.4 Grafické rozhraní

Jedním ze základních požadavků na design byla výroba grafického rozhraní pro dotykový displej. Pro tento účel .NET Micro Framework poskytuje široké množství knihoven

založených na technologii WPF[13]. Oproti plnému .NET Frameworku zde ovšem nemáme možnost psát *XAML*. Musíme tedy všechny elementy psát od píky, což znesnadňuje vývoj.

Více o grafickém rozhraní je napsáno v kapitole "Ovládání a nastavení", kde je vysvětleno ovládání a nastavení zařízení.

4.2.5 DPWS Server

DPWS neboli *Devices Profile for Web Services*. Jedná se o základní implementaci služeb pro síťovou komunikaci se zařízením na základě vyměňování standardizovaných zpráv. Implementace této služby je poskytnuta základními knihovnami .NET Micro Frameworku. Zařízení implementující tuto technologii je schopno být objeveno v síti na základě předem daného datového kontraktu. Na základě objevení pak klientovi poskytne adresy svých koncových bodů, které slouží k zasílání zpráv dle zmíněného datového kontraktu. DPWS server byl implementován za účelem umožnění uživateli ovládat zařízení na dálku a získávat nasbírané informace se záznamníkem dat na dálku. Zařízení nabízí pro komunikaci 2 koncové body:

- DeviceManagerService (sloužící k ovládání zařízení)
- LoggerService (sloužící k získávání zaznamenaných informací)

Pro umožnění vytvoření klienta je zapotřebí soubor typu *.wsdl* definující kontrakt pro komunikaci se zařízením. Pomocí tohoto souboru si můžete vygenerovat klienta, který bude s aplikací komunikovat.

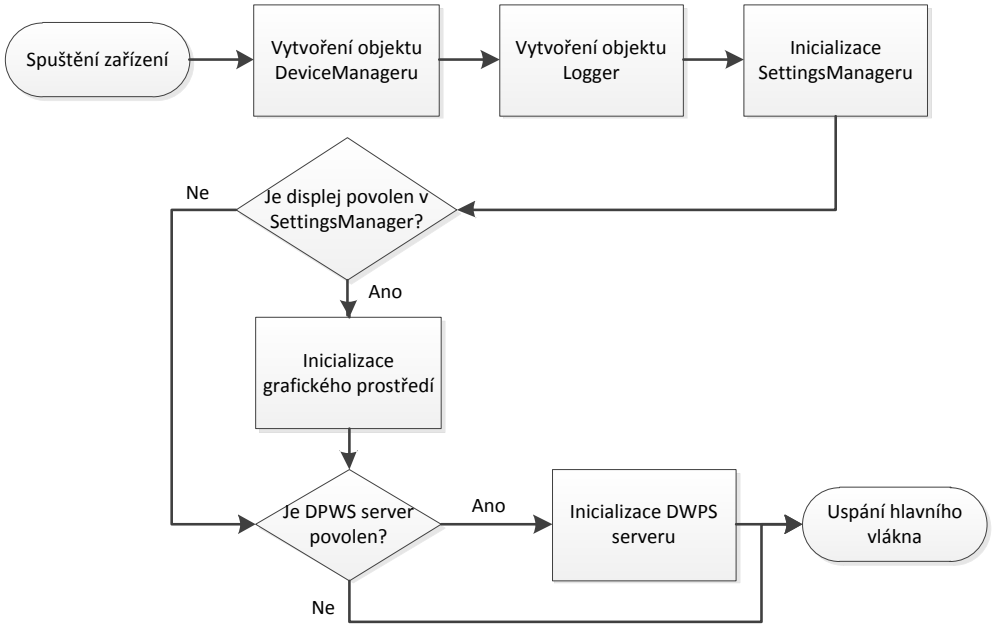
4.3 Způsob fungování aplikace

Na následujících diagramech bude nárožně ukázáno fungování aplikace pro dané typové scénáře. Ukázky budou ilustrovány na vývojových diagramech. Diagramy se týkají pouze aplikační vrstvy. Fungování na úrovni TinyCLR v nich není zmíněno.

Způsob fungování grafického rozhraní a DWPS serveru bude popsán v kapitole Nastavení a ovládání.

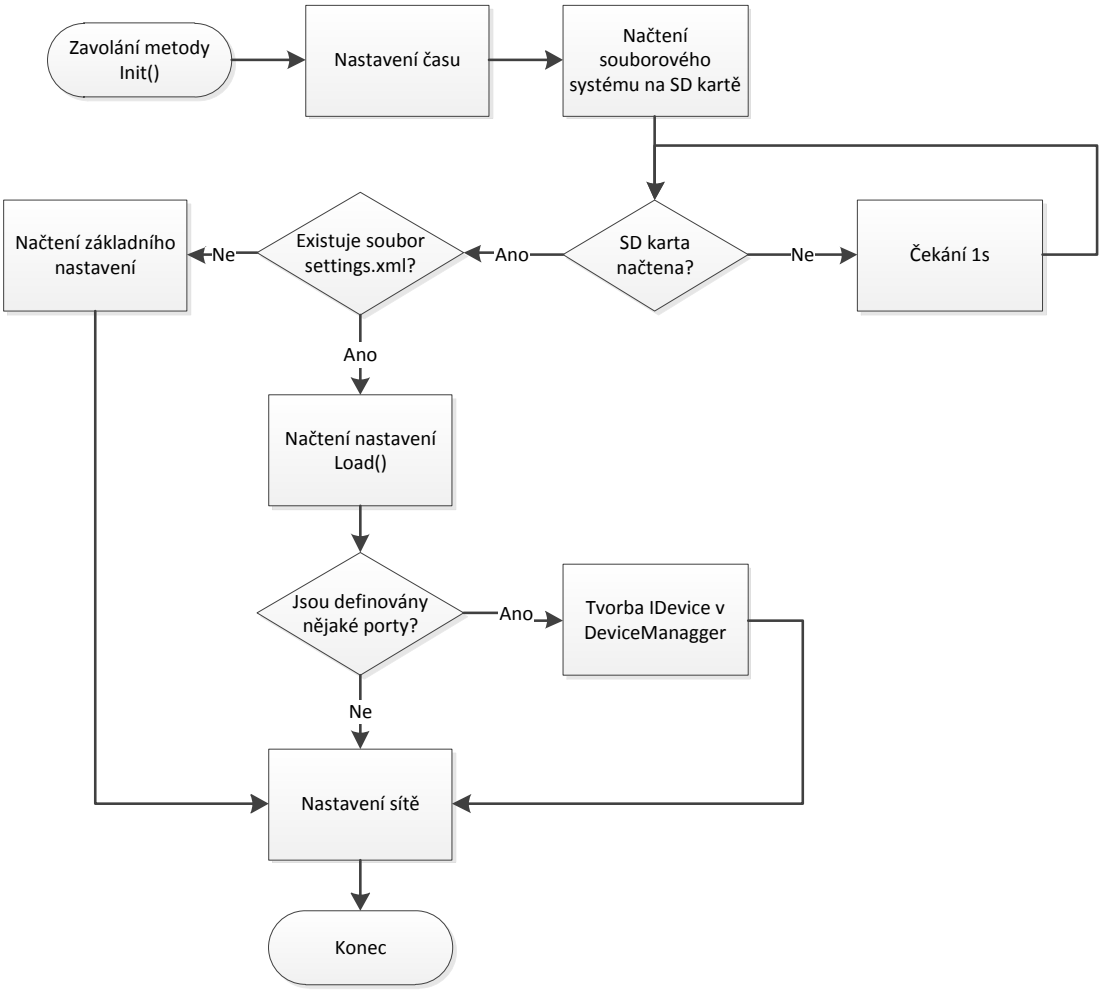
[13] Neboli Windows Presentation Foundation. Jedná se o novou koncepci vývoje grafického rozhraní, která má nahradit stávající model Windows Forms.

A) Spuštění zařízení



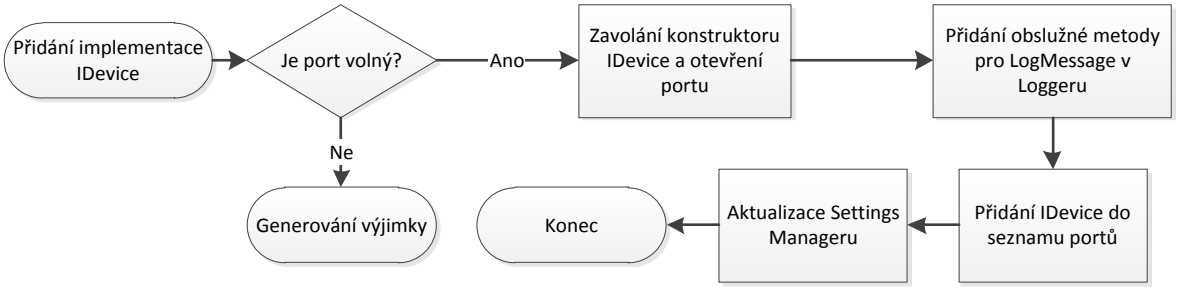
Obr. 8 Diagram pro spuštění zařízení

B) Inicializace SettingsManageru



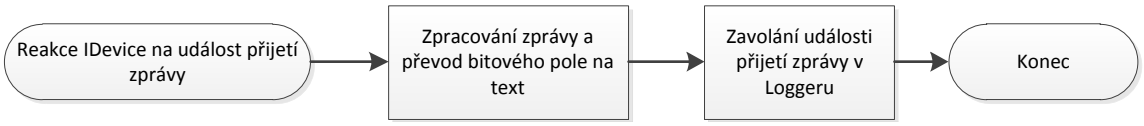
Obr. 9 Diagram inicializace SettingsManageru

C) Definice a otevření portu



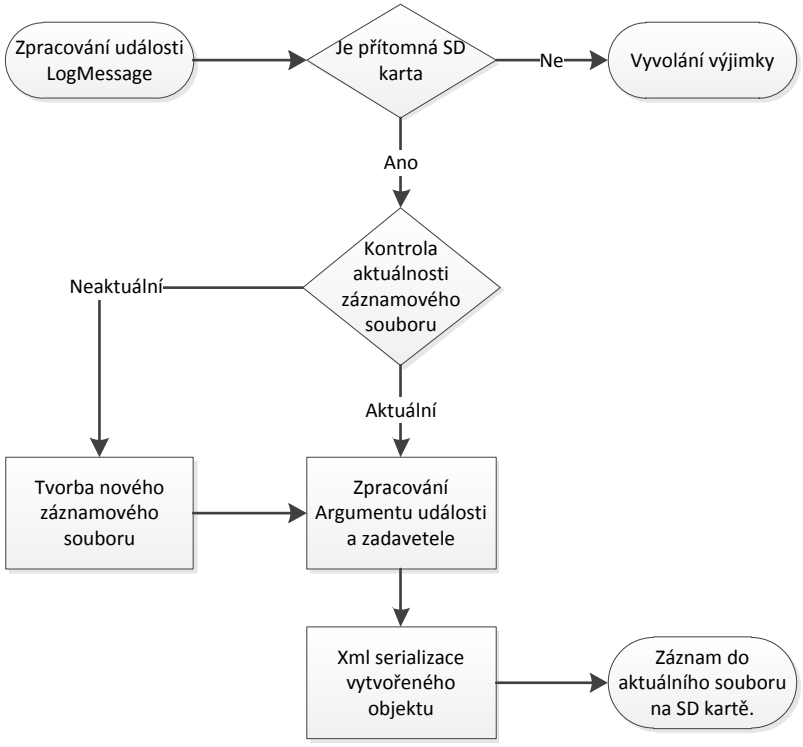
Obr. 10 Diagram přidání a definice portu

D) Přijetí datového záznamu



Obr. 11 Diagram přijetí datového záznamu

E) Zaznamenání záznamu na SD kartu v logeru



Obr. 12 Diagram zaznamenání záznamu

5 Nastavení a ovládání

Nastavení a ovládání programu lze provádět třemi způsoby:

- Definice nastavení v souboru settings.xml před spuštěním programu
- Nastavení pomocí grafického uživatelského rozhraní
- Nastavení pomocí DPWS serveru

Získání zaznamenaných dat lze uskutečnit dvěma způsoby:

- Vytažením připojené SD karty a přečtením dat přímo z ním
- Pomocí DPWS serveru

5.1 Způsoby nastavení

5.1.1 Nastavení pomocí Settings.xml

Toto nastavení je realizováno pomocí zkopírování souboru Settings.xml na SD kartu či jeho editace před spuštění zařízení. Tento způsob nastavení je zamýšlen pro zařízení, které nemají zabudovaný displej nebo ethernetový konektor.

Pokud uživatel spustí zařízení bez přítomnosti tohoto souboru na SD kartě, soubor se vytvoří v základní podobě. Do tohoto souboru jsou pak zálohována veškerá nastavení a změny provedené uživatelem v průběhu aplikace. Pokud tedy dojde k restartování aplikace, nastavení aplikace se obnoví do původního stavu.

Základní příklad pro nastavení aplikace:

```
<appSettings>
  <Appliacation DeviceON="true" ServerON="true" Network="192.168.0.200"/>
  <Devices>
    <Device Name='device name' Port='device port' Type='COMDevice' BaudRate=''
Parity='' StopBits='' Handshake='' Databits='' />
  </Devices>
</appSettings>
```

Uživatel si zde může zapnout nebo vypnout obrazovku či DPWS server pokud je nepotřebuje. Všechny změny se vždy projeví až po restartu zařízení. Existuje zde i možnost změnit nastavení IPv4 adresy zařízení. V části „Devices“ si uživatel může nadefinovat základní komunikační rozhraní, která jsou načtená po startu. V době vyhotovení této práce je možná pouze definice rozhraní typu „COMDevice“.

5.1.2 Nastavení a ovládání pomocí displeje

K vytvoření grafického uživatelského rozhraní *GUI*[14] bylo využito základních knihoven .NET Micro Frameworku. GUI se v .NET Micro Frameworku programuje pomocí technologie *WPF*[15]. Bohužel zde chybí podpora *XAML*, díky které by se vývoj uživatelského rozhraní stal přívětivějším.

[14] Zkráceně „Graphic User Interface“, neboli „grafické uživatelské rozhraní“.
[15] Zkratka pro „Windows Presentation Foundation“.

Na následujícím náhledu uživatelského rozhraní budou rozebrány všechny klíčové služby pro ovládání a nastavení zařízení. Snímek byl pro lepší kvalitu zobrazení pořízen z emulátoru.

Obr. 13 Náhled na dotykové uživatelské rozhraní

Uživatelské rozhraní je plně dotykové. Existuje zde i možnost ovládat rozhraní pomocí standardních tlačítek, která jsou k vidění na snímku zkušební desky (4 směrová a SELECT). Tato tlačítka se hodí především při nepřítomnosti dotykového displeje.

5.1.3 Nastavení a ovládání pomocí DPWS serveru

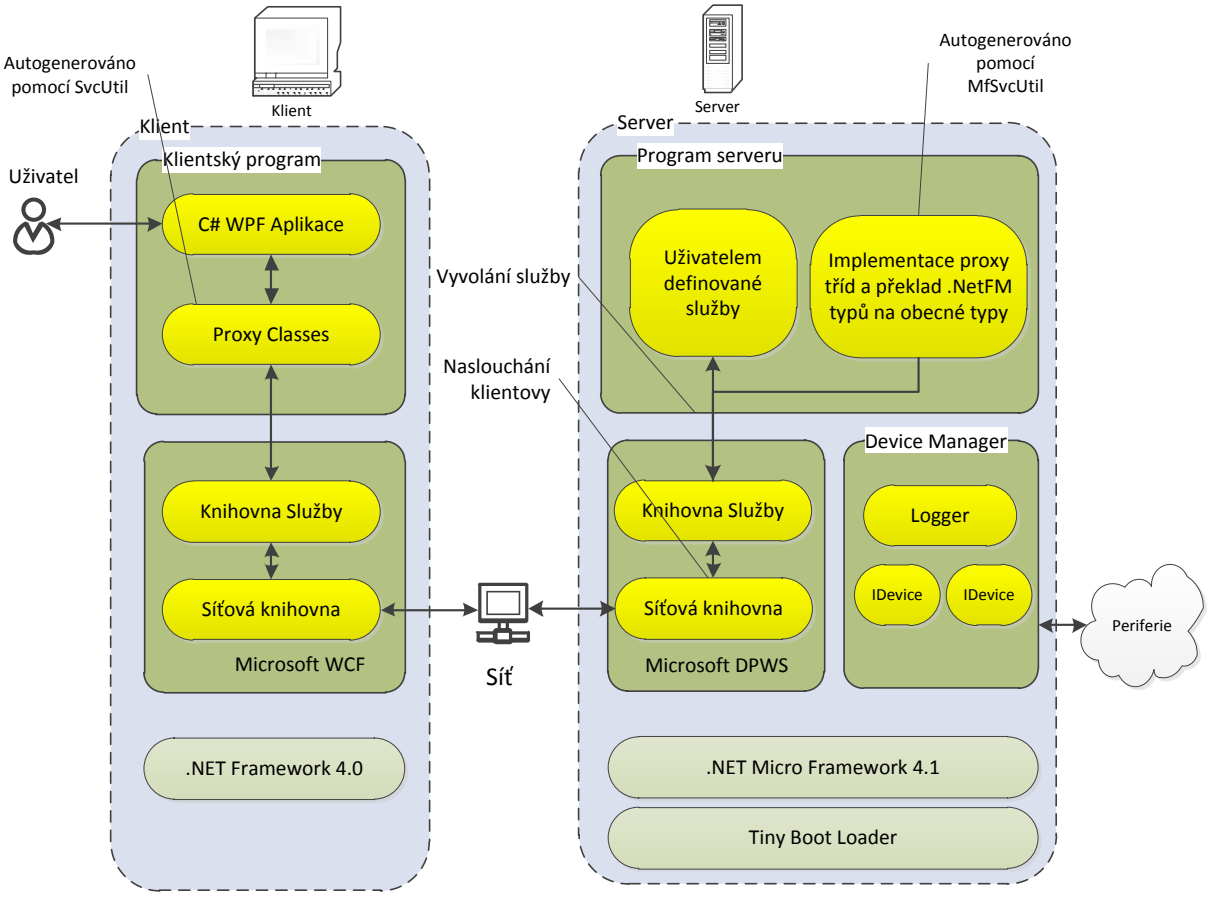
Definice DPWS serveru a poskytované služby již byly uvedeny v kapitole 4. V tomto oddílu se především zaměříme detailně na poskytované služby a znázorníme jejich fungování na schematickém diagramu.

První službou, která je nabízená skrze DPWS server je „DeviceManagerService“. Tato služba slouží ke zjišťování připojených komunikačních rozhraní a k jejich případnému nastavení či vytvoření.

Druhou dostupnou službou je „LoggerService“. Tato služba slouží k získání zaznamenaných dat skrze síť ethernet. Uživatel má možnost definovat, ze kterého časového úseku chce data zobrazit. Soubory záznamů dat jsou *Loggerem* rozdělovány po třech dnech kvůli udržení rozumné velikosti souboru.

Proxy třídy pro serializaci a deserializaci zasílaných zpráv jsou tvořeny na straně DPWS serveru pomocí nástroje MFSvcUtil, který je dostupný pro každou verzi .NET Micro Frameworku v adresáři *Tools*. Na straně klienta jsou tyto třídy tvořeny skrze nástroj SvcUtil, který je integrální součásti Microsoft Visual Studia.

Vzniklé třídy jsou poté tvořeny se stejného souboru typu *.wsdl,* který definuje datový kontrakt, jenž musí být pro klienta i server totožný.



Obr. 14 Schéma fungování DPWS serveru

6 Závěr

Programování záznamníků dat v .NET Micro Frameworku zaručilo jeho velkou rozšiřitelnost a opětovnou použitelnost vyvinutého kódu v dalších projektech. Programováním tohoto zařízení jsem získal cenné zkušenosti s jazykem C# a programování aplikací postavených na .NET Micro Frameworku. Hlavní výhodou zde byla možnost se zaměřit pouze na aplikační vrstvu, což mi podstatně usnadnilo vývoj a implementaci této aplikace.

Díky přítomnosti rozšiřitelného emulátoru jsem mohl převážnou část aplikace odzkoušet v prostředí Microsoft Visual Studia a rychle simulovat chování reálného zařízení. Funkčnost poskytnutého emulátoru je velice dobrá a kód simulovaný na emulátoru zpravidla hladce běžel i na vývojové desce EMX popsané výše v této práci. Jedinou překážkou pro testování v emulátoru byla nekompatibilita knihoven od GHI Electronics. Z tohoto důvodu bylo v projektu použito podmíněné kompilování pro emulátor.

Jednou z hlavních překážek ve vývoji je relativní mládí frameworku. Díky jeho rapidnímu vývoji je velká část dostupných materiálů neaktuální. V minulém roce s příchodem verze čtyři se situace stabilizovala a vývoj je v současnosti velice konzistentní a zpětně kompatibilní.

Složitý byl také vývoj uživatelského grafického rozhraní, což je způsobeno díky nemožnosti použít XAML. Využil jsem tedy základního grafického rozhraní, které je dodáváno se všemi zařízeními od společnosti GHI Electronics. Toto rozhraní jsem dále rozšířil a upravil podle mých potřeb.

Při vývoji byl kladen důraz na možnost rozšíření implementace komunikačních protokolů jako třeba CAN, I2C, SPI atd. Tohoto jsem docílil velkou abstrakcí kódu. Implementace dalších rozhraní by tedy měla být snadná. V době zhotovení práce je implementováno pouze zaznamenávání dat přes sériové rozhraní.

Věřím, že tento framework se bude dále rozšiřovat a upevní si své, už tak pevné místo, které má ve strategii Microsoftu pro embedded zařízení.

Seznam použité literatury

[1] Dokumentace k .NET Micro Framework SDK. MICROSOFT. *Microsoft MSDN* [online]. 2010 [cit. 2012-05-21]. Dostupné z: http://msdn.microsoft.com/en-us/library/ee436350.aspx

[2] KÜHNER, Jens. *Expert .NET Micro Framework*. 2. ed. Berkeley, CA: Apress, 2009, 481 s. ISBN 978-143-0223-870.

[3] THOMPSON, Donald a Colin MILLER. .NET Micro Framework White Paper. In: *http://msdn.microsoft.com/en-in/netframework/* [online]. 2007 [cit. 2012-05-10]. Dostupné z: http://download.microsoft.com/download/a/9/c/a9cb2192-8429-474a-aa56-534fffb5f0f1/.net%20micro%20framework%20white%20paper.doc

[4] Dokumentace k modulu EMX. In: *http://www.ghielectronics.com* [online]. 2010 [cit. 2012-05-17]. Dostupné z: http://www.ghielectronics.com/downloads/EMX/EMX_DevSys_Broch_Pinout.pdf

[5] Oficiální stránky .NET Micro Framework [online]. [cit. 2012-05-18]. Dostupné z: http://www.netmf.com/

[6] Oficiální stránky s podporou pro produkty od GHI Electronics [online]. [cit. 2012-05-10]. Dostupné z: http://www.tinyclr.com/

[7] Oficiální stránky výrobce zkušebního modulu [online]. [cit. 2012-05-23]. Dostupné z: http://www.ghielectronics.com/